View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Open Library

Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT

Information and Software Technology 45 (2003) 911–928

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

Collaboration and coordination in process-centered software development environments: a review of the literature

Paulo Barthelmess

University of Colorado at Boulder, Campus Box 430, Boulder, CO 80309-430, USA

Received 31 December 2002; revised 15 March 2003; accepted 29 April 2003

Abstract

Process-centered software development environments are systems that provide automated support for software development activities. Such environments mediate the efforts of potentially large groups of developers working on a common project. This mediation is based on runtime support for actual work performance based on formal representations of work.

In the present work, we survey and assess the contributions of the software process literature under the perspective of support for collaboration and coordination. A broad range of alternative approaches to various aspects of representation and runtime support are identified, based on the analysis of an expressive number of systems. The identified functionality can serve both as a guide for the evaluation and selection of systems of this kind as well as a roadmap for the development of new, improved systems.

© 2003 Elsevier B.V. All rights reserved.

Keywords: Process-centered software development environments; Software development support systems; Collaborative system development

1. Introduction

Software Engineering "deals with the building of software systems which are so large or so complex that they are built by a team or teams of engineers" [1]. Developing non-trivial software systems is therefore a task that requires that a group of agents work in concert, that is, they must *collaborate* in order to reach the common goal.

Process-Centered Software Development Environments (PCSDE) allow for the definition and enactment of procedures performed by groups of developers working on a common project. A PCSDE stores definitions of processes in terms of steps that need to be performed, artifacts produced and transformed by these steps, of users that should perform the steps, sometimes given in terms of roles, and of constraints on execution, such as precedence among steps.

The focus in this paper is on how capable systems are of supporting groups of software engineers in their common objective of developing systems. Two complementary aspects are key in such support: *collaboration* and *coordination*. For the purpose of this paper, we define collaboration as relating to user communication and user awareness of each other's actions; coordination is related to mechanisms that are used to avoid the need of such inter-user communication, such as division of labor and automatic distribution of work.

Large development teams are plagued by what Brooks called the 'Tar-pit' effect [2]—as team sizes grow linearly, the time spent by team members to align perspectives and to keep aware of the actions of others might grow exponentially. The challenge then is how to devise strategies for dividing the work, for assigning work to different developers and to indirectly coordinate their actions.

PCSDE tackle this problem by allowing complex work processes to be defined beforehand and by supporting actual development as processes unfold. These systems embed knowledge about processes and can serve as a source of information and guidance, thus avoiding some of the communication that would be necessary otherwise. On the other hand, while unnecessary communication should be avoided, one wants developers to be able to be as aware as possible of the work of others that might affect their own. There is a need for strong support for both indirect and direct communication among team members aiming at keeping their perspectives aligned. A support systems should to the extent possible mediate this communication.

E-mail address: barthelm@colorado.edu (P. Barthelmess).

0950-5849/03/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0950-5849(03)00091-0

Support for collaboration and coordination under this broad perspective includes a wide range of functionality. Important factors include how expressive the descriptions of work are; how effective is the distribution among team members; how flexible is the work execution; how much support is available for handling unavoidable variations, among others. Inflexible or otherwise inappropriate functionality along any of these dimensions can adversely impact the performance of a team, either because of information overload or information deprivation.

Clearly, no system can match the complexities of actual work and are therefore by definition limited (and limiting). Research in process-centered process development acknowledges this fact, even if indirectly, by proposing alternatives that make environments more useful by imposing less restrictions on the way work is performed or by better adapting to particular work styles.

The goal of this paper is therefore to examine available PCSDE literature and identify ranges of functionality along a few key dimensions. The result of this work can be read as a guide for evaluation of process support systems, by comparing offered features with those of a wide range of existing solutions. This paper can also be understood as a roadmap for developers of new PCSDEs.

For the purpose of contrasting solutions, some references to the workflow management literature are made. Workflow management systems (WFMS) are process support systems as well, but target mostly business, rather than development processes as PCSDEs. WFMS has a rich literature on flexibility and support for cooperation [3] that can sometimes shed light on some of the issues we are interested in here.

Related work deals with similar issues, but under a different focus. A survey and taxonomy of PCSDE can be found in Ref. [4]; in depth descriptions of systems' features can be found in Refs. [5–7] and in the many papers mentioned throughout this paper.

The rest of this paper is organized as follows. We start by presenting background information on the issues surrounding collaboration in the context of software development (Section 2.1). Process Centered Software Development Environments and the nomenclature used in this paper are presented in Section 2.2. We then proceed to present an analysis framework (Section 3) that is employed to classify system functionalities in this paper. The software development literature is then examined in Sections 4–8. The paper ends with summary and conclusions (Section 9). More detailed information on a some of the analyzed systems can be found in Appendix A.

2. Background

2.1. General issues in collaborative software development

In software development, from an initial abstract goal (an end state consciously selected a priori [8]) a series of incremental transformations is performed. These transformations need to be *soundness preserving*, i.e. at the end, the informational product needs somehow to match the initial goal, in difficult to qualify ways.

The objective is to keep the adherence to the goal in it's many incrementally more detailed incarnations. Each step in the process should expand the concepts of previous ones, adding detail but preserving the original intended semantics—preserving the *conceptual integrity* [2].

When the task is complex enough to require that a large team or teams be employed, keeping adherence to the goal becomes a major problem. The agents most probably have different backgrounds and perspectives that must somehow be aligned during the work, to avoid deviations from the goal. It is of the essence that all agents work in concert to achieve the desired goal throughout the transformation process. This can only be achieved by a large amount of information exchange between the involved parties [9].

Paradoxically, the same communication that is vital for maintaining the conceptual integrity introduces some of the main problems in such an effort (the Tar-Pit effect). Communication overload can easily results from the indiscriminate addition of manpower to a software development project. A key issue is, then, how to organize the effort in such a way that the appropriate level of meaningful communication is provided, but no irrelevant extra communication is necessary. The general answer here is that the resources need to be managed: routines need to be established, roles assigned, communication patterns identified and so on. In summary, an orderly *process* needs to be devised and then implemented.

The root of the problem is that each single participant of a project adds potentially up to n communication channels (where n is the number of other agents), ensuing a combinatorial explosion of communication that soon leads to overload. In other words, after a while all resources are spent in communication and none in the actual work, and of course, 'work cannot be achieved by just talking about it' [9].

Adding to the problem of communication overload is the *semantic loss* that is known to occur in longer chains of communication. The use of deep hierarchies that sound as a solution to managing and restricting the amount of intercommunication in a tree-like structure, is not a viable solution because of the degradation of information that occurs.

In summary, adherence to the original goal, the conceptual integrity, demands coupling between the agents, i.e. they have to communicate frequently to constantly realign their individual efforts. At the same time, this essential communication can reach very fast an unmanageable level that backfire on the original intent.

PCSDE can potentially help, by incorporating an abstracted description of work, allowing for distribution of work policies to be defined and controlled during actual

development. In Section 2.2 we further explore issues of automation of software development processes.

2.2. The software development process and its automation

The importance of software *processes* is directly linked to the observation, commonly accepted, that the *quality of the product* is a result of the *quality of the process* [10]. Except for the most non-critical, single person effort that does not aim at generating a product, no more than common sense is required to assess that only through an orderly sequence of steps will eventually be possible to generate a product with acceptable quality. This is not, by the way, a privilege of information system development efforts. This same statement is obviously applicable to any other non-trivial collaborative effort.

Software development presents some peculiarities, though, that makes it more vulnerable perhaps than other development efforts. There are some essential problems that make this kind of effort intrinsically hard, e.g. the high *complexity*, *conformity* and *changeability* that characterize software products [2, p. 181], plus the fact that producing software is a *creative* process, human-centered and therefore unpredictable and subject to variations. The creation of these complex entities usually takes a long time, which only worsens the pressure for change even during the creation of the software itself. After all, this same software reflects in some way a real-world need that is itself subject to changes that are unavoidable.

As a result of the afore-mentioned complexities, it is natural that the process that governs such development will itself be complex and therefore would benefit from automated support. This can be linked to Ostwerweil's observation that 'software processes are software too' [11]. In fact, the idea that similar techniques can be applied for the manufacture of software and processes is very appealing. Determining an adequate sequence of steps to be applied is also a creative, collaborative effort that results in a potentially complex object, the *process definition* or *model*, that is also subject to change. The next logical step is then to use tools to help produce, evolve and enact processes. PCSDEs are environments that provide this support for the construction, evolution and enactment of *process models*.

A *process model* is an abstract representation of software production *activities* and their relationship. Process models can be considered software objects and, as such, have *life cycles*, and are themselves specified, designed, implemented, and deployed. Work steps are the units of work, that may be sometimes combined into *Tasks* or *Activities*. Tasks can be associated to *roles*.

Roles describe in an abstract form the set of skills and/or responsibilities associated with the execution of one or more tasks. During task execution, developers create and transform *artifacts*. *Artifacts* represent the object of work in an environment, and correspond to typical software development objects, such as requirements documents, test plans, test cases, etc.

A process model usually specifies relationships among work steps or tasks, in the form of a precedence relation (e.g. compiling precedes linking). Similarly, artifact types may be related to each other, e.g. forming a hierarchy of modules and sub-modules.

Process models are meant to be instantiated, resulting in an executable entity called a *project* (or simply *process instance*). Zero or more projects based on the same or different process models can co-exist. Each instance or project can be in a different stage of its *life cycle*.

Projects are enacted (or are said to unfold) under the protection of an environment. This protection can vary widely, ranging from reporting, through guidance, to enforcement. The environment is usually responsible for keeping track of the progress of activities, their termination, and for enabling new activities as soon as their pre-conditions are met.

Enactment is guided by a *project plan*, that might initially correspond to a parameterized instance of a generic process plan that is not necessarily complete. As work progresses, project plans may be adapted to specific contingencies of a specific project, or might be complemented with additional project specific definitions. Current and historic process state and the content of artifacts produced so far by a project can influence unfolding. For instance, depending on how many sub-modules a module is defined to have at a certain moment in the development, a different number of dynamic sub-activities might be started. The complexity of artifacts therefore directly influences unfolding.

The environment is also responsible for supporting the management of the process, i.e. the *monitoring* and *adjustment* that is always necessary in face of the variability and unpredictability of software development.

Finally, the process itself is also subject to change, so *meta-processes* may be used to help in their evolution. Meta-processes may control the construction and evolution of generic process plans, or guide co-construction and evolution of specific project plans.

3. Analysis framework

Technology (including computer technology) is at the same time enabling and restricting. Tools enhance users' capabilities (e.g. to communicate), but are obviously only effective within the limits of the functionality that is made available by the tool. E-mail technology, for instance, is adequate for asynchronous support for written messages—it does not help much in cases where synchronous communication is required or desired.

PCSDEs can be seen as technology that constrains possible action according to a description of a process. The job of a process support system is to try to offer support for actual work execution based on information contained in

such a description. Constraining of work according to a description constitutes both process support systems' strength and their weakness. The strength is derived from the guidance that can be provided—only a small fraction of the possible actions lead successfully or efficiently to desired goals. By constraining what can be done, process support systems can help users focus on moving forward in the direction of a goal. Conversely, the weakness of such systems comes also from this constraining. It can be the case that actions that are necessary for actually attaining a goal in a specific situation will be outside the scope of what is allowed by a process support system. The system then becomes a hindrance, instead of a helpful tool.

In the present work, we consider collaboration support provided by PCSDEs related to five complementary aspects (Fig. 1): (1) the coverage of descriptions; (2) the latitude of interpretation; (3) user–environment interaction support; (4) support for inter-user communication; (5) support for management and assessment.

Taken as a whole, these aspects determine the overall support for collaboration and coordination offered by a PCSDE:

1. *Coverage of descriptions.* PCSDEs rely on process plans as the source of information for whatever support they offer. Such support is therefore directly linked to what can be represented in such plans. The finitude of process definition languages force them to necessarily focus on some areas, that will be made easier to deal with, while other will be difficult if not impossible.
2. *Latitude of interpretation.* The dangers of taking process descriptions to be faithful representations or models of work are well publicized. Among others, Suchman [12] and Robinson and Bannon [13] warn us of the consequences of this seemingly harmless mistake. It is now understood that workers do not (and cannot) blindly follow some strictly scripted sequence of steps. Work needs to be *situated*, i.e. adapted to actual contingencies that are many times unique to each situation. Humans draw upon their common sense, knowledge of conventions and sheer creativity to (sometimes unthinkingly) make necessary adjustments.

   Systems that constrain action to what is narrowly described in a script of some sort are unable to offer help where it is most needed, namely, when some unpredicted situation surfaces.
3. *User–environment interaction.* The way systems expose their services to users is of course key in determining their usability. Different paradigms facilitate different aspects of work. User interaction paradigms usually trade off flexibility for guidance.
4. *Inter-user communication.* It is known that an important part of group work is dedicated to realignment of individual views, particularly in presence of breakdowns. An essential part of work is thus performed collectively, rather than in isolation. How much support a system offers for such communication, and how seamless is the interaction of this functionality may have therefore a high impact on system usability for collaboration.
5. *Management and assessment.* Transparency of the collective work is an important issue in project development. PCSDEs have a potential to make managerial measurements available and provide tools for inspecting the state projects are in at any time. Such functionality provides the means for managers and team leaders to keep track of progress and intervene whenever they find fit, corresponding to the human aspect of coordination.

Process description

Latitude of description interpretation
- Enforcement policy
- Evolution and deviations

User-environment interaction
- Interaction paradigm
- Binding strategies

Inter-user communication
- Communication through artifact sharing
- Integration of group collaboration tools
- Built in synchronous communication support

Management and assessment
- Monitoring
- Assessment/Measurement
- Planning/Resource management

Fig. 1. Analysis framework.

The analysis that follows is based on published literature describing systems' functionalities. References to systems are made throughout the text, to illustrate features and capabilities. The main cited systems are described in Appendix A, that can therefore be read before the analysis, in preparation for the discussion, or afterwards, to obtain a consolidated picture of capabilities related to individual systems.

4. Process descriptions

Process descriptions ultimately drive process-oriented systems' execution. The coverage that is provided therefore directly impacts system usability. Process models usually cover a variety of aspects, such as control, artifacts, tools and user roles. Different styles of specification result from focus on one of these aspects that is taken to be central. A few interesting specification style alternatives are explored in PCSDEs.

Rule-based specifications are employed by the majority of the analyzed PCSDE (e.g. MARVEL [14], OIKOS [15],

EPOS [16], Merlin [17] to mention a few). Rules usually specify an event causing their activation, a guarding *condition* and an *action* to be taken in case the condition is true. Another popular rule format surrounds an action with pre- and post-conditions.

Rules permit a *proscriptive* style of specification [18]. Proscriptive specifications allow freedom of performance—any sequence of actual operations are acceptable, provided they result in transformations in which the constraints (expressed as rules) hold.

The downside is a potential lack of guidance and harder analysis that might result from their 'openness'. Heimbigner [18] highlights the fact that a proscriptive style would better match human psychology, by not being overly constraining.

Task-based or *step-directed* specifications are the ones in which processes are expressed as a partial order of work steps or tasks, based on ordering constraints (e.g. step A must be executed before step B).

These specifications are usually associated with (1) activity-graph representations, e.g. Petri nets used in SPADE [19] or (2) procedural code, as e.g. in APPL/A [20].

Petri nets are bipartite directed graphs composed of *places* p and *transitions* t connected by *arcs* C.

Places p connected to a transition t by arcs $C(p,t)$ are said to be t's *input places*. Conversely, places connected to a transition t by arcs $C(t,p)$ are t's *output places*.

Places can contain zero or more *tokens*. Transitions whose input places contain tokens are said to be *enabled*. Enabled transitions can *fire*, which is usually associated with the occurrence of some event. The firing of a transition causes tokens to be removed from all input places, and inserted into its output places.

The state of a Petri net is giving by its *marking*, which is the set of tokens that are in specific places at any moment of time.

This elegant formalism can be used to specify concurrency and synchronism and for this reason is popular for the specification of processes.

Both graph and procedural representations may lead to a *prescriptive* style of specification. A prescriptive style specifies in detail allowed actions, usually as a sequential series of instructions that must be followed. Fully prescriptive specifications may be a hindrance at run-time. Since it is not usually possible to anticipate all possible variations of a process, specially a collaborative one, there is a high chance that performance will at some point deviate from the fully prescriptive specification. On the other hand, prescriptive specifications are easier to analyze, as a consequence of the restrictions imposed on the space of possible performances.

Even systems that could be considered step-based, as SPADE and APPL/A integrate rule-based functionality. APPL/A uses triggers and predicates associated to relations to handle constraints [20]; SPADE's Petri net transitions are associated with conditions and actions, so that the net rather than describe activity precedence, specifies the precedence among rules. In both cases the paradigm is better described as being mixed.

It is interesting to notice that in contrast, most WFMS are graph-oriented and therefore task-based.

Artifact-based modeling is centered on the objects that need to be produced, rather than on the tasks that produce them.

In artifact-based representations, operations are attached directly to artifact definitions. Transformations performed on artifacts through operations may trigger notifications and cause automatic actions to be performed, e.g. compilation after editing is completed, as in Shamus [21].

This style of specification is semantically equivalent to a task-based one in the sense that artifacts, operations and their precedence can be specified using both styles. What changes is the focus of life-cycle descriptions, that shift from an activity centered view to an artifact transformation one.

Artifacts can be related to each other, forming e.g. hierarchies, as proposed by PROSYT [22], where repositories, folders and sub-folders can be used. Operations in PROSYT can be attached to individual artifacts and also containers (repository or folder).

Role-based specifications, as the name implies, are centered around roles that are played by executors at enactment time.

Yu and Mylopoulos [23] present an *actor dependency* model that centers on actors and their relationships. An assessment of the model in the context of a large software maintenance organization is presented in Ref. [24].

Cain and Coplien [25] introduce *Pasteur*, a process evaluation framework that is also centered on roles.

In SOCCA [26], human interaction is incorporated in the model. Role behavior can be modeled (through state transition diagrams) along with other modeling entities (e.g. artifacts, operations). Synchronization between these entities is then specified in PARADIGM, a formalism to coordinate parallel processes.

The style of specification and the latitude of interpretation are closely connected. That is what is examined in Section 5.

5. Latitude of description interpretation

An old and powerful insight of the software process community is that the focus in a process should not be on modeling some abstract representation of work, but rather on understanding and supporting the dynamic and contingent way an actual process unfolds in use [5].

Collaborative processes are characterized by the impossibility of completely pre-defining their unfolding due to the high degree of change and potential breakdowns that are known to occur. The main component of collaborative work is therefore dedicated to *articulation* [27]. A key issue in a system that aims at supporting

collaboration is therefore the flexibility allowed at run-time to support such varied and unpredictable unfolding.

In this section, we consider three aspects of unfolding: (1) Enforcement policy (how strictly performance needs to match a plan), (2) Evolution (changing a plan to align it to performance) and (3) living with inconsistencies (allowing deviations from a plan).

5.1. Enforcement policy

Many PCSDE position themselves as guidance tools [28], rather than enforcers. Even though it may be questioned to what degree these systems really guide rather than enforce [28, p. 337], it is clear that there is a general agreement in the software process community that strict enforcement is not a desirable property of a process support system, and that environments 'only provide the infrastructure within which creative and cooperative work is performed' [29].

Interesting alternatives to enforcement exist:

- At one end of the spectrum, there are systems that just track the progress as users go about doing their work, e.g. Provence [30], that separates process enactment from process execution, in order to enhance the degree of non-intrusiveness [4,31]. The assumption is that users know what to do without being explicitly told and that therefore the system can remain most of the time invisible. Interaction with the users takes the form of notifications (e.g. email messages) that are sent to users to make them aware of upcoming activities.
- Other alternatives involve making visible actions that are possible at each moment, allowing them to be executed in whatever order is desired by the users, provided that constraints are respected.

  This kind of support is usually associated with systems that employ an artifact-based style of interaction (Section 6.1), e.g. Merlin [17], PROSYT [32], Shamus [21].
- A goal-based strategy affords users additional flexibility and support. In systems such as MARVEL [14], EPOS [16], Grapple [33] and ALF [34], users are free to choose any possible action supported by the system, independently of pre-conditions being enabled.

  These systems use inferencing to dynamically build plans that cause the pre-condition of an action to be satisfied so that it can be executed. Dynamic plans take into consideration actual state of a project and can thus describes a broader range of possible alternative unfoldings than is usually possible in purely prescriptive, topological models [35].

  Operations can therefore be read as *goals* that users want to achieve, perhaps in many steps, if necessary. Once the necessary preceding actions are identified, forward chaining causes their automatic execution, if possible; non-automatic actions are directed to the user.

  While backward chaining (inferencing) allows great flexibility in choosing what to do, forward chaining produces a sequence of steps that need to be followed in order to reach a goal. The strategy therefore mixes the best of both worlds—freedom of execution plus guidance through more complex process steps if required.

5.2. Evolution and beyond

It is known that static process descriptions do not match well actual performance [12,36]. This is particularly true in collaborative processes, due to their exploratory nature. Divergence between plans and actual performance is therefore frequent.

Two strategies can be employed in presence of deviations from a plan. The traditional approach, both in PCSDE and WFMS is to force realignment through plan evolution, i.e. by modifying a plan to make it match the contingency found at execution time. The second approach proposes that deviations be tolerated. We elaborate briefly on each of the approaches in Sections 5.2.1 and 5.2.2.

5.2.1. Process evolution

The method of choice for evolution in PCSDE (and to some extent in WFMS) is *reflectivity*. Processes are modified within processes (or more precisely, meta-processes). Unfolding is then a result of intertwined execution of a process, that coordinates work, and a meta-process, that coordinates changes to the way work is conducted.

- In SPADE [19] plans can be used as tokens, that in this system are associated to artifacts subjected to transformations according to a (meta) process.
- In EPOS [16] any process element, including plans, can be used as parameters to processes. EPOS explicitly supports meta-processes as part of the system. The creation and evolution of process models themselves is therefore an integral part of the system.
- ALF [34] supports the instantiation of partially defined processes. During enactment, missing parts of a plan can be instantiated taking into consideration the history of the process so far.

5.2.2. Process deviation

As pointed out by Cugola [32], the effort required to change a project plan makes this approach unsuitable to coping with situations that require minor or temporary deviations. Instead of requiring alignment of a plan every time a deviation occurs, some researchers [32,37–40] propose tolerating deviations and allowing inconsistencies to exist. In other words, performances can explicitly diverge from plans.

The bulk of the literature on deviation tolerant models is focused on inconsistent states in artifacts, a problem that is particularly relevant in software engineering, where

intricate relationships between artifacts usually exist. Inconsistencies that are directly process related, such as divergence in the order of execution, execution by some user different from the one anticipated and so on, are addressed eg. in Refs [32,38].

In Cugola [32], a dynamically established *deviation handling policy* determines classes of constraints that can be violated. The dynamic aspect allows one to establish different policies for different process phases, or for different users—an expert may be trusted to violate more constraints than a novice user.

Borgida and Murata [38] propose reifying activities and workflows, storing related information in classes that are accessible at execution time, e.g. ordering of tasks and constraints. Constraint violations are flagged and handling can be performed either by an automated handler or by users, by modifying the reified information (e.g. changing the order of steps). Deviations can be tolerated through the use of *excuses*, objects that record, e.g. the authorizing agent, reason for deviation and so on.

6. User–environment interaction

Two main aspects determine the interaction between process environments and users: the *interaction paradigm* (what the interaction is centered on) and the *user binding strategy*. *Interaction paradigm* refers to the main entity around which interaction revolves, e.g. steps, artifacts, goals or roles. *User binding strategy* is the one employed to match users to work that has to be performed.

6.1. Interaction paradigm

Interaction paradigms are usually tightly connected to the style of specification (Section 4). According to Bandinelli et al. [41,42], support for the varied requirements of interaction requires decoupling the interaction model from the semantics of the process modeling language (PML). In other words, the interaction style can (and should) be separated from the modeling style. It is usually the case that the interaction mode offered directly reflects the internal model used in the modeling language (exceptions being e.g. SPADE, Merlin and OIKOS, see below).

PCSDEs offer a rich set of alternative approaches to the human–environment interaction problem:

- *Step based*: Systems such as MARVEL [14], Oz [43] and ALF [34] offer command interfaces through which users select actions to execute.

  The paradigm employed is therefore similar to shell-based operations (e.g. in Unix) familiar to many programmers. On the one hand this potentially offers the utmost flexibility and freedom of action, but on the other hand this style provides the least amount of guidance.

  In systems that support backward and forward chaining, e.g. MARVEL [14], EPOS [16], Grapple [33] and ALF [34], actions may cause a system to dynamically produce and initiate execution of a plan that satisfies the pre-conditions of a chosen action, thus greatly enhancing the usefulness of this style (Section 5.1).
- *Artifact based*: Systems such as PROSYT [32] and Shamus [21] center performance on artifacts and operations upon them.

  This style is usually associated to some form of graphical representation of artifacts, that allows for selection of operations to be performed by clicking on representative icons or menus associated to artifacts. Automatic actions are transparently executed as artifacts change state as operations are applied.
- *Virtual environment based*: Systems adopt a paradigm originally employed by certain collaborative games (Multi-User Dungeons, MUDs). The paradigm is based on interconnected *rooms*, each containing *objects* that can be carried around and acted upon. Doppke et al. [44] discuss possible different mappings between process concepts and those employed in MUDs. Each 'room' can be made to correspond to a task, or to a person's workspace, or to an artifact or finally to some resource, such as a testing laboratory.

Some systems offer a mixed strategy:

- Process Weaver [45] presents users with both actions and related documents that should be used or produced. These are encapsulated in *working contexts* that are placed into users' *agendas* (a sort of 'inbox').
- Merlin [17], despite supporting a rule-based style of specification, employs a mixed role/artifact style of interaction. It presents in graphical form the relevant documents available for each role, as well as their inter-relationship. Operations are made available through menus. Merlin 'moves' the documents to the next work context as the work assigned to each role is completed.
- ALF [34] also defines working contexts for each role. Within working contexts are artifacts that can be shared by multiple users playing the same role.
- SPADE [19] allows the use of any interaction style, at the cost of programming done in SLANG, the Petri net based formalism used by the systems, that can be used to control external tools in detail.

Note that these alternative modes of interaction may be more adequate in situations where users work in a smaller number of projects or processes, as opposed to a production situation in which they may receive a high volume of requests of potentially disparate processes for which they have to provide some shorter duration service (as is typical, e.g. in production-oriented WFMS).

6.2. The role of humans: binding strategies

Binding has to do with the mechanism that is employed to match executors to what needs to be performed at any one time. At one end of the spectrum, strictly *proactive* systems decide when tasks should be started and which user they should be routed to. At the other end of the spectrum, users may be totally in charge of initiating tasks whenever they find fit. *Reactive* systems, e.g. Provence [31] assume that users know what they are doing and simply track user activity, mapping it into a project plan, remaining invisible most of the time.

Whenever a proactive policy is followed by a system, there must be some way of binding users to (non-automatic) tasks as they get ready to be scheduled, as opposed to reactive enactment policies, in which users initiate tasks themselves and become automatically responsible for their execution.

A mixed strategy combines reaction to user initiated actions with proactive firing of derivable actions. A cascade of such automatic activations can take place, until a decision point is reached, in which case the system stops firing derived actions and waits for further human initiated actions. Such systems display a combination of *reactive* (user initiated) and *proactive* (system initiated) actions. Goal-based systems such as MARVEL [14] and ALF [34] provide a mixed strategy (Section 5.1).

Alternative binding mechanisms are supported by some systems. Selection of users based on their attributes (e.g. availability, special skills) is proposed in Refs. [46,47]. In SPADE users may be selected according to 'ownership' of some artifact, e.g. a module being coded. If, for instance, some interface that is used by many modules is changed, the system is able to request that the person responsible for these potentially affected modules perform an evaluation of the proposed interface changes [41].

7. Inter-user communication

In conventional WFMS, collaboration between users is usually restricted to asynchronous sharing of artifacts, usually forms or documents, i.e. artifacts are handled in a serial way, from task to task. In particular, there is usually no support for concurrent access to documents in parallel tasks. Such collaboration through document hand-off is *lossy* [48]—much of the knowledge is *not* communicated in this type of transfer. Documents represent just the end result of a potentially complex chain of knowledge acquisition actions that were employed to construct them. Hence, reliance on this type of channel is only viable in situations where very little additional knowledge is required, other than the directly implied by the document itself.

PCSDE approach the issue of user–user interaction in three different ways: (1) by supporting synchronous sharing of artifacts, (2) by integrating collaboration tools into the environment, and (3) by providing built in synergistic group support. We next examine each of these approaches.

7.1. Communication through artifact sharing

The understanding that users in a collaborative process share artifacts (such as documents, code) is deeply rooted in PCSDE [49]. The mechanisms employed are versioning and extended transaction models [50,51]. Functionality at this level is usually tied to the underlying support provided by a database where artifacts are stored. In general, the mechanisms make users aware of possible conflicts and help them resolve such conflicts.

Detailed presentation of the complex issues involved is beyond the scope of the present paper. Interested readers are referred to Refs. [51,52].

- In ALF [34], several agents can share a single role, which allows them to work in the same working context, therefore sharing information—different contexts can share artifact instances.
- OIKOS [15] offers a rich set of metaphors that handle different levels of co-operation: the *desk*, *environment* and *office*. Desks can be shared informally by a group of cooperating users.
- In Adele/Tempo users operate on private working environments that contain copies of artifacts stored in a versioned database (Adele). A transaction manager handles conflicts that occur when more than one user modifies the same artifact in parallel.
- EPOS [16] builds on a database (EPOS-DB) that offers versioned, long, nested and cooperating (non-serializable) transactions. Access conflict resolution can be specified in detail through a specific language (WUDL: the Workspace Unit Declaration Language).
- Merlin builds on the GEMSTONE database and offers long transactions. It can employ pessimistic and optimistic strategies. Users can help resolve conflicts by choosing a solution given transactional information (e.g. who is holding a lock).
- Shamus [21] keeps users aware of each others actions, in an attempt to reduce conflicts.

Collaboration through concurrency control, even though essential whenever concurrent access to the same artifacts exists, does not offer the synergy that is necessary in real collaboration. Concurrency control, offers only a *reactive*, after-the-fact kind of collaboration [53]. Other enhanced levels of collaboration are offered through the mechanisms that we briefly described in Section 7.2.

7.2. Integration of group collaboration tools

An enhanced level of human collaboration support is provided by integrating PCSDE with external tools that are able to handle a closer mode of collaboration [4]. Usually,

the tool that is integrated adds some form of synchronous communication support, such as support for electronic meetings.

In some systems, external synchronous communication tools are incorporated through the use of an existing tool integration functionality. Such integration is done, e.g. in SPADE [41], that controls the ImagineDesk toolkit from inside a process. A similar approach is presented in Ref. [54], where the Marvel PCSDE [14] is integrated with ConverationBuilder [55]. In Ref. [56], Merlin [17] is integrated with Multiview, a group collaboration tool to visualize multiple artifact representations.

An issue regarding tool integration is how closely coupled this integration really is [41,53]. Ideally, the integrated tool can be made to influence the process and vice-versa, i.e. there is a close coupling that permit the process to control the tool and that actions performed by users of the tool be visible to the process, so that it can react to them accordingly.

7.3. Built in synchronous communication support

Some systems make an option of directly supporting synchronous cooperation. Oz [53], for instance, offers such built in support, that works in combination with the support for tasks executed by groups of users. Serendipity [57] supports synchronous operations through filter/actions, that can be made to coordinate the use of synchronous tools. The process model itself can be synchronously edited by a group of collaborating users.

Other systems support synchronous communication as a consequence of the paradigm chosen, typically based on some kind of *virtual space* or *environment*, as the one employed in MUDs [44] and locales [58]. Space-based paradigms can support synchronous communication in a natural way, by making visible the 'presence' of other users that are 'visiting' a room at the same time (Section 6.1).

8. Management and assessment

Management control aspects take form of *meta-activities*, activities whose focus of attention is not the development of software itself, but the way by which software is developed. In other words, their object is the work itself, not the product of the work.

Change seems to be inherent to the development of software, due to the fact that development is also (or primarily) a discovery process. At start little is known about the problem, and as work progresses, the increased understanding may cause different approaches to be more desirable.

The amount of success of a PCSDE depends heavily, as a consequence, on how well the meta-activities are supported by the environment. Meta-activities are managerial in nature and include *monitoring*, *assessing*, *planning* and *adjustment*.

8.1. Monitoring

By *monitoring* we mean the managerial activities that involve checking the progress of work against some schedule, to proactively verify if some adjustment needs to be done.

Few references can be found about this topic in the literature. Conceivably, monitoring could be attained through the modeling of the appropriate role, but this is in general not discussed in the literature.

- In Process Weaver, an *Owner's View* allows a project manager to overview processes in execution. State of activities and documents can be visualized, as well as the overall state of a process, expressed as mappings on a Petri net [59].
- In PADM, it is suggested that the agents themselves provide the managers with 'time-sheets' with the necessary information [60].
- Merlin allows managers to retrieve on-line information about project status [56]. This functionality takes advantage of the system's backward chaining capabilities to respond to queries about the process state, represented as a dynamically updated Prolog-like rule-base.
- Provence keeps a graphical representation of the state of processes. This representation can be examined through a generic visualization tool—Dotty [30].

8.2. Assessment/measurement

A prerequisite for process improvement is the ability to measure objectively the performance of current processes. This data can then be used to inform the improvement and fine tuning of the process in a continuous way (as proposed, e.g. by the CMM [61]).

Data collection can be incorporated into most process models through appropriate language constructs (that vary from system to system), but no standard for that is available. A few systems offer data collection and assessment functionality:

- In Process Weaver, a *tracer* tool can be used to collect statistical information on process execution, like the number of times some activity is executed, or the number of times a loop is iterated [59].
- SynerVision [59] keeps (as task attributes) some measurement information, e.g. estimated time for completion and actual time spent. Other attributes can be defined to fit organization specific metrics.
- Endeavors can be customized to interact with metric gathering tools [62, p. 22].

Another aspect mentioned by some researchers [63,64] has to do with the fact that it seems to be specially difficult to collect exact measurements, given the fact that some of the relevant actions are performed outside

the scope of the environment, e.g. in phone conversations, meetings, etc (off-line). This makes it difficult to maintain the internal representation of the state with the real one. As one of the consequences, measuring the process is not realistic if only the events captured by the system are considered.

8.3. Planning/resource management

The creation of processes is a managerial task that precedes in time the process itself [65]. Existing process models and process instances (projects) are also subject to (re)planning, whenever some change forces the way the work is done to be modified.

Realistic managerial planning includes not only the planning of process steps, but also *resource* management, like the assignment of activities, determining their deadlines, determining costs and so on [66]. Little is mentioned in the literature regarding this aspect. According to Huff, this is one of the least developed aspects of process models [63].

- Some references are made in Ref. [66] to modeling of these aspects in Statemate.
- Effort, time and budget are considered resources in PEACE [28].
- In PADM, it is suggested the integration of an external scheduling support tool using the usual tool integration facility of the system [60].
- Process Weaver [45] can be integrated to existing project management tools. Library functions can be used to periodically scan a project management tool's database. Activities that are due to get started are then automatically instantiated by Process Weaver.
- Endeavors allows the augmentation of the system with project management software. Traditional project management resources (budgets, computers, meeting rooms, people) are considered part of the model [62].

9. Summary and conclusions

PCSDE features were analyzed in the context of collaboration. PCSDE are by nature geared towards collaboration and coordination, due to their focus on supporting software development, an activity that can be characterized as creative, exploratory collaborative work.

Some features offered by such systems can be useful in the general case:

- Range of possible enforcement policies, from tracking of user initiated actions to strict, push based enforcement.
- Rich notion of interaction paradigms that include modes such as *step based*, *artifact based*, or based on *virtual environment* paradigms. Mixed strategies are also employed.
- Support for multi-user tasks. Selection of users, when automatic, can be based on user attributes, such as ownership of objects or skills.
- Sharing of artifacts in parallel tasks is supported through enhanced concurrency mechanisms and versioning.
- Synchronous communication is supported either through tool integration or as a built in functionality.
- Specifications have broad scope, and include support for finer grained sub-processes. Artifact directed unfolding is provided.
- Evolution is not mandatory—systems can live with inconsistencies.

On the downside, broader coverage for collaboration and coordination comes at the cost of more complex process descriptions. Languages are usually too complex for users, even if these users are software developers themselves. The effort required to build and evolve a process can be too high.

Modeling languages are in general low-level, and biased towards the implementation phase and may not be appropriate for all phases of development, particularly the more creative design phases [4].

One key aspect is that technology by itself is unable to provide a final answer to the problem of collaboration. There are intrinsic limits to how much help tools can provide. Tools can at most provide adequate support for knowledge representations. Building and using such knowledge can only be done by humans. The payoff of using a tool is therefore intrinsically limited.

Acknowledgements

The contribution of the anonymous reviewers is gratefully acknowledged. Their thorough and insightful comments helped to substantially improve the presentation of this paper.

Appendix A. Overview of the systems

In this section, PCSDEs are described in fuller detail, complementing the analysis presented in the previous sections. Further reviews of these and other systems can be found, e.g. in Ambriola et al.'s excellent survey [4] and in Refs. [5–7].

A.1. Adele/Tempo

Adele/Tempo [67,68] was developed at IMAG in Grenoble.

Adele is a versioned database in which process components are stored. Process steps are modeled as objects that define operations, attributes, and recursively, other process steps. User activities are modeled as methods associated with processes and attributes.

Tempo is a rule-based process modeling language. Tempo is based on Event-Condition–Action rules (ECA) extended with time, i.e. the event part can include temporal expressions. Rules are used to control the initiation of activities.

Users perform their activities through *working environments*. These are Adele sub-databases associated with one user. Working environments are composed of directories and files, tools and a task to be performed [7]. A *process engine* monitors execution according to defined rules, e.g. guiding compilation and testing of modules.

A *transaction manager* handles conflicts that might occur when shared objects are accessed concurrently. Configuration management in Adele is based on the *branch* concept. A branch models a sequence of revisions, that are snapshots of the object attributes.

A.2. ALF

ALF [34] defines models in a language called MASP/DL (Model for Assisted Software Process Data Language). Process models consist of hierarchies of model fragments, called MASP. Each MASP is defined by a 5-tuple (Om, OPm, Rm, ORm, C) where

- *Om* is an *object model* that describes data used in the fragment;
- *OPm* is a set of *operator types* which are abstractions of tools or tool families, e.g. *edit*, *compile*, that are later associated with actual tools;
- *Rm* is a set of Event-Condition–Action (ECA) *rules* expressing reaction to events, e.g. triggering of a linker after successful compilation takes place;
- *ORm* is a set of *ordering constraints* that express how operators can be applied (e.g. *edit*(*x*); *compile*(*x*) denote that edits must precede compilations);
- *C* is an invariant, called the *characteristics* that must be true at all times for a specific fragment.

The set of operator types (OPm) define pre- and post-conditions for operator execution. The activation of an operator whose pre-conditions have not yet been satisfied causes a plan to be automatically built through inferencing. This plan tries to fulfill the violated pre-condition. If that fails, the user is notified, otherwise the plan is carried out and the operation is executed.[1] Similarly, the 'characteristics' C causes a similar reparation plan to be built, whenever it becomes false as a result of changes in the state of a fragment.

MASPs are recursively structured by defining operators as MASPs. An operation of a higher level fragment can thus correspond to a whole other fragment.

ALF supports the iterative construction of process models while in execution, by instantiation of MASPs that have undefined operators. Later in the process, these operators can be instantiated according, e.g. to the history of process enactment.

User interaction with the system happens through a few tools, e.g. an *action tool* that allows users to select operations they want to execute; a *guidance tool* that analyzes the impact of an action, or shows what can be done next; a *reporting tool* that shows what has been done and how these results were obtained.

ALF uses a virtual file system that emulates the Unix file system (PCTE) to map transparently the file operation requests made by tools into their database equivalents.

A.3. APPL/A

APPL/A [20] is a process programming language used in the Arcadia Project [69].

Process descriptions in APPL/A are procedural, written using the basic mechanisms and constructs provided by the Ada programming language. APPL/A extends Ada to include constructs that support process modeling, in the form of first class *relations* and associated services (e.g. *triggers*, transactional statements).

Relations connect software elements in a data model based on extensions of Codd's relational model. They can be used to explicitly represent interdependencies among software elements.

Triggers add a rule-based flavor to the language and are used, e.g. for change propagation, log maintenance and reactive services in general. Predicates can be used to express constraints on the state of relations.

Serializability of relations and optional recoverable access is achieved through extended transactional statements. These constructs can be used to implement, e.g. conventional, nested and cooperative transactions.

A.4. EPOS

EPOS [16,70–73] (Expert System for Program and ('og') System Development) was developed at the University of Trondheim, Norway.

Models are expressed in SPELL, a concurrent reflective language that is a superset of Prolog. Different aspects of a process are defined in sub-models: *activity/task model*, *product model*, *tool model*, *human and role model*, *cooperation model and meta-process model* [70]. Pre-conditions relate tasks to each other. Task models therefore form *task networks*.

SPELL supports a mix of specification styles, ranging from rule-based to a task-based style using scripts (surrounded by pre- and post-conditions) [4, p. 291] (Section 4). Process elements are uniformly modeled as SPELL types and instances, including process plans themselves.

EPOS supports meta-processes used for the creation and evolution of process models as an integral part of the system.

[1] This mechanism is thus similar to the one in MARVEL (Section A.5) and EPOS (Section A.4).

A reflective object-oriented language can be used to specify policies for model creation, composition, change, instantiation, refinement and enaction [71]. SPELL tasks can handle any process element as parameters, thus allowing for process plans themselves to be the objects of (meta) processes.

Two components: the *execution manager* and the *planner* work in tandem to handle enactment. The execution manager evaluates task pre-conditions. Instances whose conditions evaluate to true are executed directly if they are atomic. Composite tasks cause the execution manager to invoke the planner. The planner automatically generates a new sub-task network, using artificial intelligence (AI) planning technology. The post-condition of the composite task is taken to represent a *goal*. The planner applies backward chaining and hierarchical decomposition to build an appropriate sub-network that corresponds to a plan in AI terms. Since sub-networks are built dynamically and incrementally, the system automatically readjusts to changes to task definitions (types) and product structure. The planner can be automatically invoked whenever changes are introduced [73].

EPOS-DB is the component that is responsible for artifact storage in this system. EPOS-DB stores both process description and data on which processes operate, as well as the relationships among these elements [7]. EPOS-DB offers versioned, long, nested and cooperating (non-serializable) transactions with mutual cooperation protocols [16]. The versioning mechanism is based on change-oriented versioning. In this model, the *deltas* (a set of changes) are kept as separate entities that are applied selectively to baseline artifacts. The combination of a baseline artifact and one or more change sets originates a logical version. Predicates in version-rules dictate the change-sets that should be applied.

A language—Workspace Unit Declaration Language (WUDL)—can be used to specify how access conflicts should be handled, e.g. breaking of locks, controlling change propagation and version merging [4, p. 300].

A.5. MARVEL

MARVEL [14,74,75] is a project conducted at Columbia University by Kaiser and colleagues.

Processes are specified in MARVEL through rules expressed in a notation called the MARVEL Strategy Language (MSL). Three sets are specified for a process: *rules*, *types* and *tools*. These define, respectively, (1) the process specific issues, (2) objects employed, and (3) tools that operate on the objects [7].

Rules consist of a *name* that corresponds to a user command, a list of *object types* used as parameters, a *precondition* for initiation, an *action* expressed as a tool envelope (a Unix shell script) and one or more *effects* (post-conditions) [75, p. 18]. One post-condition is chosen from the set of available one, depending on a code returned by the tool that was activated as part of the action, e.g. success or failure in compiling a piece of code.

MARVEL employs forward and backward chaining capabilities to support users in identifying and performing actions that need to be carried out in order to satisfy the pre-conditions of some other action. The activation of a rule whose pre-conditions are not yet satisfied causes the system to apply backward chaining to identify other rules (and associated actions) necessary to fulfill these pre-conditions (unless the rule explicitly forbids this behavior [17, p. 23]). Forward chaining is then applied to these identified rules, causing actions to be carried out automatically whenever possible, or sent to a user for manual execution [7].

Users select the commands they want to execute. No enforcement is thus imposed (Section 5.1). As a result of the pre-condition satisfying logic described above, users may be proactively requested to execute additional actions. There are no facilities for associating users to specific steps of a process. Such facilities were later added to Oz, a successor of MARVEL (Section A.8).

An experiment integrating MARVEL and Converation-Builder [55] (a group collaboration tool) is reported in Ref. [54].

A.6. Merlin

Merlin [17,56,76] was developed at the University of Dortmund, as part of a project of the same name.

Modeled elements include *activities*, *roles*, e.g. programmer, manager, *software objects* (artifacts) and *resources* (people participating in the project). Software objects are associated to activities which transform objects. Inter-object relationships are also represented. Activities in turn are associated to tools supporting them. Groups of activities can be associated to roles. Roles, in turn, are associated to users (resources) that can play them.

Process representation is based on rules stored in a shared process database. Prolog-like backward chaining rules are used to select roles and activities a user may perform, and to answer queries on the process state. Forward-chaining rules describe proactive responses generated by the system and consist of a pre-condition, list of activities and post-condition. Relationships among system objects are stored as a dynamic persistent graph structure [56]. Rules and relationships are updated during process execution, thus reflecting process change and unfolding in a flexible way. Rules reflect both a process structure as well as facts about executing instances, e.g. marking the operations already performed on a code module [76], in a Prolog-like style.

User interaction is centered in *working contexts* presented for each role. The working context presents the set of objects (represented as boxes) that are

associated with the role along with their dependencies (represented as arrows connecting the boxes), along with the activities that can be performed on each object (as menus attached to the boxes). Appropriate tools are automatically launched and controlled by the system as a response to a user selecting an action to execute. Users can select to hide some of the objects on their working contexts (e.g. specifications). The style of interaction is therefore artifact-based (Section 6.1), but organized under a role perspective/view.

Cooperation is supported in an indirect way by the environment, that 'moves' documents between the working contexts of the different roles. In other words, when some document has been successfully modified by one role, it disappears from this role's context and is included in the working context of the role responsible for the next step of transformations. Once a piece of code is completed by a programmer, it may be moved automatically to the working context of a quality assurance engineer. If the tests performed by the latter result in a failure, the piece of code might, for instance, be moved back to a programmer's context for correction [76].

Users can choose to execute available actions in whichever order they prefer. The system proactively executes automatic actions that become enable as a result of completion of other actions. Guidance can be proactively supported by forward-chaining rules, that might take a user through the steps required, e.g. for testing some code [7].

Transactional support is built on top of services offered by GEMSTONE, an OODBMS. Both long transactions (called *working context transactions*) and short ones (*activity transactions*) are supported. Optimistic and pessimistic strategies can be employed. Users are presented with transaction information (who is holding which lock) and may help choosing the right type of transaction.

An experiment in tool integration produced a version that incorporates Multiview, an Integrated Development Environment. Multiview supports multiple representations of code in addition to the conventional textual form, for instance, graphical representations of code fragments. This integration illustrates the system's capability of incorporating and controlling complex tools and is described in Ref. [56].

A.7. OIKOS

OIKOS [4,15,77] gets its name from the ancient Greek word for *house*, that has acquired a meaning related to *environment* (as, e.g. in *ecology*). This system was developed in the late 1980s at the Università a di Pisa by Montangero and colleagues.

Limbo is the system's language used to specify processes. Paté is an enactment language (actually a sub-language of Limbo). Both languages are declarative. These are logic languages that are compiled into an intermediate language that is interpreted by a pre-defined set of Prolog programs. The system employs blackboards/tuple spaces to handle concurrency and distribution. Execution is driven by rule-based agents that fire non-deterministically and concurrently, based on pre-conditions that are unified with blackboard tuples. The body of a rule is a Prolog program. Post-conditions specify tuples that are to be written as a result of the execution of the body.

OIKOS offers a rich set of paradigms that handle different levels of cooperation, the *desk*, *environment* and *office*. A desk corresponds to a shared work space, potentially used by different roles to share information about the state of their work. Several agents may play their roles on the same desk, cooperating in a free fashion. Desks are part of *environments*. Several groups can play their roles in different desks, but under the same environment. The environment controls access to shared documents and therefore allows communication formalized by the environment's rules. Finally, an office groups many environments. The interaction of groups in different environments is the most formalized one and is controlled by the surrounding process.

Users interact with the system through graphical user interfaces that expose the contents of the blackboards as icons. Users then apply commands directly to these icons. This style of interaction can be considered *artifact based* (Section 6.1).

A.8. Oz

Oz [43,46,53,78] was developed at Columbia University as an enhancement to the MARVEL system (Section A.5).

Oz supports geographically dispersed teams by coordinating a federation of decentralized autonomous processes [43]. These autonomous processes are described using a common formalism (as opposed to systems such as Process-Wall [79] that supports multiple formalisms).

Oz shares the process description language of its predecessor, MARVEL. MARVEL's rule specification is extended to include the definition of executors. In Oz, activities associated to rules can have multiple executors. The selection of users is based on a flexible query mechanism that can be used, for instance, to identify all users that are connected to a document that needs to be changed by an *owner* relationship. As a result, changes to different documents will be executed by potentially different groups of people [53].

Oz embeds support for synchronous group collaboration through which a group of users appointed as executors of an activity can jointly perform it. Such group activities might be supported, for instance, by group editors, or shared white-boards [53].

A.9. Process weaver

Process weaver [45,80] is a commercial PCSDE.

It defines process models as (1) a hierarchy of activity

types (denoted *methods*) and (2) an associated flow control specification (called *cooperative procedure*).

Methods decompose higher level activities into lower-level ones, in a tree-like structure. Each activity is associated with a textual description, input and output artifact types and the human roles involved.

Activities are represented in cooperative procedures as Petri net transitions. Transitions are associated with a condition and an action. Conditions and actions can be expressed in the Co-shell language, which is similar to Unix shell languages. A few pre-specified conditions can also be employed: wait for event generated by one or a group of users (through workcontexts buttons, discussed below); wait for a specific place to be marked in another cooperative procedure (allowing synchronization); or empty (no condition). Similarly, pre-defined actions can be used: send a workcontext to an agent or to a group of agents; initiate the execution of another cooperative procedure; or empty (no action).

Workcontexts correspond to individual work assignments, associated to atomic work-steps in a process. Workcontexts group a set of documents, tools, help references and buttons. Buttons generate corresponding events that are then used in conditions in cooperative procedures, as discussed above.

Users interact with the system through their individual *agendas*, which display the workcontexts each one is supposed to work on. 'Send to group' actions cause copies of a workcontext to be placed in the agendas of all users in a group. The corresponding 'wait for event generated by group' will only become true when all users in a group have signaled an event through their individual workcontexts.

Process Weaver can be integrated to project management tools. Through a library of standard procedures, a project management repository can be periodically scanned and activities that are due to start can be automatically instantiated by Process Weaver.

A *tracer* tool can be used to collect statistical information on process execution, like the number of times some activity is executed, or the number of times a loop is iterated [59].

A.10. PROSYT

PROSYT [22,32] (PROcess Support sYstem capable of Tolerating deviations) was developed at the Politecnico di Milano by Cugola and colleagues, based on previous experiences related to the SPADE system (Section A.13) and to SENTINEL [81].

PROSYT is based on a distributed event infrastructure, and offers code mobility to support nomadic users. It is built on top of the JEDI infrastructure, developed locally to provide necessary services for the system [22].

Process specification is artifact-based (Section 4). Artifacts can be organized hierarchically by placing them into *repositories* and *folders* within repositories. Folders in turn can contain a mixture of other folders and artifacts.

Repositories, folders and artifacts (that we will call *system objects*) can be associated with *operations* that can be directly invoked by users and *automatic operations* that are proactively executed by the system. Automatic operations are triggered by events, e.g. the invocation of some operation in some other related artifact.

Constraints and invariants can be set for system objects. Constraints provide guidance as to which actions can be chosen by users, but are not mandatory. Users can override the constraints and execute actions independently of the condition of the guarding expressions. The system keeps track of these deviations and guarantees that invariants are satisfied.

As indicated by the name of the system, the main research focus is on tolerating deviations (Section 5.2). A dynamically established *deviation handling policy* determines classes of constraints that can be violated. The dynamic aspect allows one to establish different policies for different process phases, or for different users—an expert, for instance, may be trusted to violate more constraints than a novice user.

The style of specification is therefore proscriptive (Section 4), with the added benefit of flexible overriding of constraints (Section 5.2).

A.11. Provence

Provence focus on providing non-intrusive support for project execution. In this system, users go about their jobs as usual, activating regular tools they are used to, such as their favorite editors. Provence transparently monitors users' actions and maps them back into corresponding process actions. A project's state can thus be kept updated with little or no direct intervention from users (users still have to inform the system the completion of activities such as meetings, that might not be represented by any action taken through a computerized system). As a result of this state update, notifications can be generated, advising users of upcoming activities.

The system is built from a combination of four general purpose tools: a *process server* (MARVEL), a *smart file system* (the 3D File System), an *event engine* (Yeast), and a *graph visualization tool* (Dotty). A component called the *enactor* binds these tools together. The enactor is responsible for mapping actual user actions into equivalent process model activities (interfacing the event engine and the process server). The enactor also updates a network representation of the process state that is used by the graph visualization tool.

The event engine is initially loaded with a description of the low level file system events to be monitored, that would indicate that corresponding process activities are taking place. These low level events are extracted from a process model kept by MARVEL.

The smart file system traps user file-related actions and reports them back to the event engine, that in turn causes

updates to a project state, via a mapping performed by the enactor.

The main question is of course how a system such as this can determine in which phase the artifacts that are transparently controlled are. A programmer may, for instance, open and save a piece of code repeatedly in the course of many days while she is working on it. The meaningful event would in this case be not the repeated opening and closing of a file, but something else that indicates that the code is considered to be ready. The solution seems to be related to the use of folders. Whenever an artifact moves to a new state, it is required that the users write them to a different (apparently pre-specified) folder. It is this file writing action that is then taken to represent completion of an activity [30].

A.12. Shamus

Lamarca et al.'s Shamus System [21] explores a document-centered approach to software development support. Shamus is an 'application' within the Placeless Documents system. The approach used consists of attaching *active properties* to documents, in the specific case of Shamus, to code fragments. Active properties can be used, for instance, to 'implement access control, handle reading and writing from repositories and to perform notifications of document changes to interested parties' [21, p. 6].

Development is supported by providing awareness of changes others are applying to a code base in real-time, aiming at reducing conflicts when code is checked in back to repositories. The system can show, for instance, which methods and classes are being modified even before the respective files are checked back in. Other active properties allow for the automation of common development tasks such as compilation and code generation whenever code is changed.

The approach is therefore *artifact based*, both from the point of view of specification (Section 4) as from the point of view of interaction (Section 6.1). The system employs a mixed reactive/proactive strategy (Section 6.2): on the one hand users are responsible for choosing whatever action is available at any moment, in any desired order, on the other hand, the system can proactively initiate automatic actions, e.g. the above-mentioned compilation and document generation.

A.13. SPADE

SPADE [19,41,82,83] was developed in the early 1990s by Bandinelli and colleagues at the Politecnico di Milano.

Its specification language (SLANG) is based on a higher-level Petri Net formalism. Petri net transitions are associated to Condition–Action rules. Enabled transitions wait for corresponding conditions to become true. When that happens, an action is initiated and tokens are written to the transitions' output places. Initiation of a compilation might, for instance, cause a token to be written to a place that enables a linking rule. The linking rule's condition waits for compilation to complete before activating the linker. Actions are specified using a logic-like language. The topology of the net describes a precedence relation among rules.

Project artifacts are embedded in objects that correspond to the tokens flowing in a net. These tokens/objects are managed by an object-oriented database (O_2). Objects are transparently moved to/from the database to a file system to allow the use of conventional (non-database-enabled) tools to operate on them.

Rules associated to transitions are responsible for extracting an enabling token from the database as part of their condition. Rules also include a set of statements that specify how output tuples are created [4].

In summary, SLANG combines in a single formalism both the object-oriented representation of artifacts (data aspect) as well as the specification of the synchronization and parallelism through a Petri net (control aspect).

The formalism is used to uniformly represent different aspects of a process, e.g. human resource management, interaction between users and the environment (describing complex sequences of tool activations, for example), in addition to the basic specification of events and their precedence relation.

The flexibility of SLANG is explored in the system to integrate tools, for instance, based on the DEC FUSE tool integration suite [84], and Sun Tooltalk [85]. Through SLANG, tools can be controlled in detail, and results of tool execution can be reified and made to flow as tokens. More sophisticated integrations were also implemented, e.g. the detailed control of the ImagineDesk toolkit from inside a process [41]. ImagineDesk is a group conferencing system also developed at the Politecnico di Milano. The resulting integrated system bridges two models of collaboration, the synchronous collaboration provided by the conferencing system and the asynchronous process support provided by SPADE.

Process evolution is supported by a reflective mechanism: process definitions themselves can be seen as tokens. As such, process definitions can be manipulated in meta-processes that apply some transformation, as would happen to any other artifact/object flowing through a net in the system.

Users interact with SPADE through tools, some of which are conventional software development tools and others are specific SPADE tools, such as the process agenda that provides feedback on the state of a project. The presence of the environment is therefore kept most of the time hidden from users. Its presence is mostly felt indirectly, as user-initiated actions cause related actions to take place. Different styles of interaction (Section 6.1) can be supported at the cost of specifying them in SLANG [42].

References

[1] C. Ghezzi, M. Jazayeri, D. Mandrioli, Fundamentals of Software Engineering, Prentice Hall, Englewood Cliffs, NJ, 1991.

[2] F. Brooks, The Mythical Man-Month: Essays on Software Engineering, Anniversary ed., Addison Wesley, Reading, MA, 1995.

[3] M. Klein, C. Dellarocas, A. Bernstein (Eds.), Computer Supported Cooperative Work (CSCW), The Journal of Collaborative Computing, vol. 9, Kluwer, Dordrecht, 2000, (special issue on Adaptive Workflow Systems).

[4] V. Ambriola, R. Conradi, A. Fuggetta, Assessing process-centered software engineering environments, ACM Transactions on Software Engineering and Methodology 6 (3) (1997) 283–328.

[5] A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.), Software Process Modelling and Technology, Research Studies Press Ltd, Taunton, Somerset, England, 1994.

[6] A. Fuggeta, A. Wolf (Eds.), Trends in Software: Software Process, Wiley, New York, 1996.

[7] P. Armenise, S. Bandinelli, C. Ghezzi, A. Morzenti, Software process representation languages: survey and assessment, in: Fourth International Conference on Software Engineering and Knowledge Engineering, Capri, Italy, 1992, pp. 455–462.

[8] Principia Cybernetica Web, Web Dictionary of Cybernetics and Systems, 1998. Available on the web at http://pespmc1.vuc.ac.be/ASC/indexASC.html

[9] J. Bardram, Designing for the dynamics of cooperative work activities, in: Conference on Computer-Supported Cooperative Work, ACM, 1998, pp. 89–98.

[10] R. Conradi, C. Fernström, A. Fuggetta, Concepts for evolving software processes, in: A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.), Software Process Modelling and Technology, Research Studies Press Ltd, Taunton, Somerset, England, 1994, pp. 9–31, (Chapter 2).

[11] L. Osterweil, Software processes are software too, in: W.E. Riddle (Ed.), Proceedings of the Ninth International Conference on Software Engineering, IEEE Computer Society Press, Monterey, CA, 1987, pp. 2–13.

[12] L. Suchman, Plans and Situated Actions: The Problem of Human–Machine Communication, Cambridge University Press, Cambridge, UK, 1987.

[13] M. Robinson, L. Bannon, Questioning representations, in: Proceedings of the Second European Conference on Computer Supported Cooperative Work, Amsterdam, Netherlands, 1991, pp. 219–233.

[14] N.S. Barghouti, Supporting cooperation in the marvel process-centered sde, in: H. Weber (Ed.), Fifth ACM SIGSOFT Symposium on Software Development Environments, Tyson's Corner VA, vol. 17, 1992, pp. 21–31, (special issue of Software Engineering Notes).

[15] C. Montangero, V. Ambriola, Oikos: constructing process-centred sdes, in: A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.), Software Process Modelling and Technology, Research Studies Press Ltd, Taunton, Somerset, England, 1994, pp. 131–152, (Chapter 6).

[16] R. Conradi, Epos: object-oriented cooperative process modeling, in: A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.), Software Process Modelling and Technology, Research Studies Press Ltd, Taunton, Somerset, England, 1994, pp. 33–70, (Chapter 3).

[17] G. Junkerman, B. Peuschel, W. Schäfer, S. Wolf, Merlin: supporting cooperation in software development through a knowledge-based environment, in: A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.), Software Process Modelling and Technology, Research Studies Press Ltd, Taunton, Somerset, England, 1994, pp. 103–130, (Chapter 5).

[18] D. Heimbigner, Proscription versus prescription in process-centered environments, in: T. Katayama (Ed.), Proceedings of the Sixth International Software Workshop, ISPW6, IEEE, Hakodate, Japan, 1990, pp. 99–102.

[19] S. Bandinelli, A. Fuggetta, C. Ghezzi, L. Lavazza, Spade: an environment for software process analysis, design and enactment, in: A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.), Software Process Modelling and Technology, Research Studies Press Ltd, Taunton, Somerset, England, 1994, pp. 223–248, (Chapter 9).

[20] S. Sutton, D. Heimbigner, L. Osterweil, Language constructs for managing change in process-centered environments, in: Proceedings of the Fourth Symposium on Practical Software Development Environments, 1990, pp. 206–217.

[21] A. Lamarca, W.K. Edwards, P. Dourish, J. Lamping, I. Smith, J. Thornton, Taking the work out of work flow: mechanisms for document-centered collaboration, in: European Conference on Computer Supported Cooperative Work, ECSCW'99, 1999, pp. 1–20.

[22] G. Cugola, C. Ghezzi, Design and implementation of prosyt: a distributed process support system, in: IEEE Eighth International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Palo Alto, California, 1999, p. 32.

[23] E. Yu, An actor dependency model of organizational work—with application to business process reengineering, in: Proceedings of the On Organizational Computing Systems (COOCS'93), Milpitas, California, 1993, pp. 258–268.

[24] L. Briand, W. Melo, C. Seaman, V. Basili, Characterizing and assessing a large-scale software maintenance organization, in: D. Perry (Ed.), Proceedings of the 17th International Conference on Software Engineering, ACM Press, Seattle, Washington, 1995, pp. 133–143.

[25] B.G. Cain, J.O. Coplien, A role-based empirical process modeling environment, in: Proceedings of the Second International Conference on the Software Process: Continuous Software Process Improvement, Berlin, Germany, 1993, pp. 125–133.

[26] G. Engels, L. Groenewegen, Socca: specifications of coordinated and cooperative activities, in: A. Fuggeta, A. Wolf (Eds.), Trends in Software: Software Process, Wiley, New York, 1996, pp. 71–102.

[27] E.M. Gerson, S.L. Star, Analyzing due process in the workplace, ACM Transactions on Information Systems 4 (3) (1986) 257.

[28] J. Lonchamp, An assessment exercise, in: A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.), Software Process Modelling and Technology, Research Studies Press Ltd, Taunton, Somerset, England, 1994, pp. 335–356, (Chapter 13).

[29] C. Ghezzi, Introduction and summary of the workshop, in: C. Ghezzi (Ed.), Proceedings of the Ninth International Software Process Workshop (ISPW9'94), IEEE Computer Society Press, Airlie, Virginia, 1994, pp. vii–ix.

[30] N.S. Barghouti, B. Krishnamurthy, Provence: a process visualization and enactment environment, in: Proceedings of the Fourth European Software Engineering Conference, ESEC'93, Springer-Verlag, Garmisch-Partenkirchen, Germany, 1993, pp. 451–465.

[31] N.S. Barghouti, Separating process model enactment from process execution in provence, in: C. Ghezzi (Ed.), Proceedings of the Ninth International Software Process Workshop (ISPW9'94), IEEE Computer Society Press, Airlie, Virginia, 1994, pp. 70–73.

[32] G. Cugola, Tolerating deviations in process support systems via flexible enactment of process models, IEEE Transactions on Software Engineering: Special Section on Managing Inconsistency in Software Development 24 (11) (1998) 906–907.

[33] K. Huff, V. Lesser, A plan-based intelligent assistant that supports the software development process, in: P. Henderson (Ed.), Proceedings of the ACM Symposium on Practical Software Development Environments, 1988, pp. 97–106.

[34] G. Canals, N. Boudjlida, J. Derniame, C. Godart, J. Lonchamp, Alf: a framework for building process-centered software engineering environments, in: A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.), Software Process Modelling and Technology, Research Studies Press Ltd, Taunton, Somerset, England, 1994, pp. 153–186, (Chapter 7).

[35] G. Kaiser, S. Popovich, I. Ben-Shaul, A bi-level language for software process modeling, in: W. Tichy (Ed.), Trends in Software, Wiley, New York, 1994, pp. 39–72, (Configuration Management issue; Chapter 2).

[36] H. Saastamoinen, On handling exceptions in information systems, Jyvskyl Studies in Computer Science, Economics and Statistics, University of Jyvskyl, 1995.
[37] R. Balzer, Tolerating inconsistency, in: L. Belady, D. Barstow, K. Torii (Eds.), Proceedings of the 13th International Conference on Software Engineering, IEEE Computer Society Press, Austin, TX, 1991, pp. 158–165.
[38] A. Borgida, T. Murata, Tolerating exceptions in work flows: a unified framework for data and processes, in: Proceedings of International Joint Conference on Work Activities Coordination and Collaboration, WACC'99, San Francisco, CA, 1999, pp. 59–68.
[39] K. Narayanaswamy, N. Goldman, 'Lazy' consistency: a basis for cooperative software development, in: Proceedings of the ACM Conference on Computer Supported Cooperative Work, 1992, pp. 257–264.
[40] B. Nuseibeh, To be *and* not to be: on managing inconsistency in software development, in: Proceedings of the Eighth International Workshop on Software Specification and Design (IWSSD8'96), 1996, pp. 164–169.
[41] S. Bandinelli, E. Di Nitto, A. Fuggetta, Supporting cooperation in the spade-1 environment, IEEE Transactions on Software Engineering 22 (12) (1996).
[42] S. Bandinelli, E.D. Nitto, A. Fuggetta, L. Lavazza, Coupled vs. decoupled user interaction environments in psees, in: C. Ghezzi (Ed.), Proceedings of the Ninth International Software Process Workshop (ISPW9'94), IEEE Computer Society Press, Airlie, Virginia, 1994, pp. 50–52.
[43] I. Ben-Shaul, G. Kaiser, A paradigm for decentralized process modeling and its realization in the oz environment, in: B. Fadini, L. Osterweil, A. van Lamsweerde (Eds.), 16th International Conference on Software Engineering, IEEE Computer Society Press, Monterey, CA, 1994, pp. 179–188.
[44] J.C. Doppke, D. Heimbigner, A.L. Wolf, Software process modeling and execution within virtual environments, ACM Transactions on Software Engineering and Methodology 7 (1) (1998) 1–40.
[45] C. Fernstrom, Process weaver: adding process support to Unix, in: Proceedings of the Second International Conference on the Software Process, IEEE CS Press, 1993, pp. 12–26.
[46] I. Ben-Shaul, G. Kaiser, Integrating groupware activities into workflow management systems, in: Seventh Israeli Conference on Computer Systems and Software Engineering, IEEE Computer Society Press, Monterey, CA, 1996, pp. 140–149.
[47] V. Gruhn, Communication support in a process-centered software engineering environment, in: C. Ghezzi (Ed.), Proceedings of the Ninth International Software Process Workshop (ISPW9'94), IEEE Computer Society Press, Airlie, Virginia, 1994, pp. 37–41.
[48] D.E. Perry, Issues in process architecture, in: C. Ghezzi (Ed.), Proceedings of the Ninth International Software Process Workshop (ISPW9'94), IEEE Computer Society Press, Airlie, Virginia, 1994, pp. 138–140.
[49] B. Warboys, D. Balasubramaniam, R. Greenwood, G. Kirby, K. Mayes, R. Morrison, D. Munro, Collaboration and composition: issues for a second generation process language, in: O. Nierstrasz, M. Lemoine (Eds.), Proceedings of the European Software Engineering Conference, ESEC'99 no. 1687, Lecture Notes in Computer Science, Springer, Touluse, France, 1999, pp. 75–91.
[50] A. Elmagarmid (Eds.), Transaction Models for Advanced Database Applications, Morgan Kaufmann, Los Altos, CA, 1992.
[51] N.S. Barghouti, G.E. Kaiser, Concurrency control in advanced database applications, ACM Computing Surveys 23 (3) (1991) 269–317.
[52] D. Tombros, A survey of database support for process centered software development environments, Technical Report 95.28, Universität Zürich, 1995.
[53] I.Z. Ben-Shaul, G.T. Heineman, S.S. Popovich, P.D. Skopp, A.Z. Tong, G. Valetto, Integrating groupware and process technologies in the oz environment, in: C. Ghezzi (Ed.), Proceedings of the Ninth International Software Process Workshop (ISPW9'94), IEEE Computer Society Press, Airlie, Virginia, 1994, pp. 114–116.
[54] J.E. Arnold, Toward collaborative software processes, in: C. Ghezzi (Ed.), Proceedings of the Ninth International Software Process Workshop (ISPW9'94), IEEE Computer Society Press, Airlie, Virginia, 1994, pp. 107–109.
[55] S.M. Kaplan, W.J. Tolone, A.M. Carroll, D.P. Bogia, C. Bignoli, Supporting collaborative software development with conversation builder, in: ProcecToward collaborative software proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments, 1992.
[56] C.D. Marlin, B. Peuschel, M.J. McCarthy, J.G. Harvey, Multiview-merlin: an experiment in tool integration, in: Proceedings on 1993 Software Engineering Environments Conference, Reading, England, 1993, pp. 35–48.
[57] J. Grundy, J. Hosking, Serendipity: integrated environment support for process modelling, enactment and work coordination, Automated Software Engineering: Special Issue on Process Technology 5 (1) (1998) 27–60.
[58] S.M. Kaplan, C. Bignoli, W.J. Tolone, Process, space and software development support, in: C. Ghezzi (Ed.), Proceedings of the Ninth International Software Process Workshop (ISPW9'94), IEEE Computer Society Press, Airlie, Virginia, 1994, pp. 100–103.
[59] P. Garg, M. Jazayeri, Process-centered software engineering environments: a grand tour, in: A. Fuggeta, A. Wolf (Eds.), Trends in Software: Software Process, Wiley, New York, 1996, (Chapter 2).
[60] R. Bruynooghe, R. Greenwood, I. Robertson, J. Sa, Warboys, Padm: towards a total process modelling system, in: A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.), Software Process Modelling and Technology, Research Studies Press Ltd, Taunton, Somerset, England, 1994, pp. 293–334, (Chapter 12).
[61] M. Paulk, B. Curtis, M. Chrissis, C. Weber, Capability maturity model, version 1.1, IEEE Software 10 (4) (1993) 18–27.
[62] G. Bolcer, Flexible and customizable workflow execution on the www, PhD Thesis, University of California, Irvine (1998).
[63] K. Huff, Software process modeling, in: A. Fuggeta, A. Wolf (Eds.), Trends in Software: Software Process, Wiley, New York, 1996, (Chapter 1).
[64] A. Wolf, D. Rosenblum, Process-centered environments (only) support environment-centered processes, in: Proceedings of the Eighth International Software Process Workshop (ISPW8), IEEE, Wadern, Germany, 1993, pp. 148–149.
[65] R. Snowdon, B. Warboys, An introduction to process-centered environments, in: A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.), Software Process Modelling and Technology, Research Studies Press Ltd, Taunton, Somerset, England, 1994, pp. 1–7, (Chapter 1).
[66] B. Curtis, M. Kellner, J. Over, Process modeling, Communications of the ACM 35 (9) (1992) 75–90.
[67] N. Belkhatir, J. Estublier, W. Melo, Adele-tempo: an environment to support process modeling and enaction, in: A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.), Software Process Modelling and Technology, Research Studies Press Ltd, Taunton, Somerset, England, 1994, pp. 187–222, (Chapter 8).
[68] N. Belkhatir, W. Melo, The need for a cooperative model: the adele/tempo experience, in: C. Ghezzi (Ed.), Proceedings of the Ninth International Software Process Workshop (ISPW9'94), IEEE Computer Society Press, Airlie, Virginia, 1994, pp. 90–92.
[69] R. Kadia, Issues encountered in building a flexible software development environment: lessons from the arcadia project, in: Proceedings of ACM SIGSOFT Fifth Symposium on Software Development Environments, Tyson's Corner, VA, 1992, pp. 169–180.
[70] M. Nguyen, A. Wang, R. Conradi, Total software process model evolution in epos, in: W.R. Adrion (Ed.), Proceedings of the 19th International Conference on Software Engineering, ACM Press, Boston, 1997, pp. 390–399.

[71] M. Jaccheri, R. Conradi, Techniques for process model evolution in epos, IEEE Transactions on Software Engineering 19 (12) (1993) 1145–1156.
[72] M. Nguyen, R. Conradi, Workspace management: supporting cooperative work, Technical Report, Norwegian Institute of Technology (NTH), Trondheim, Norway (April 1993).
[73] C. Liu, R. Conradi, Automatic replanning of task networks for process model evolution in epos, in: I. Sommerville (Ed.), Proceedings of the Fourth European Software Engineering Conference (ESEC'93), 1993.
[74] G. Heineman, G.E. Kaiser, N.S. Barghouti, I. Ben-Shaul, Rule chaining in marvel: dynamic binding of parameters, IEEE Intelligent Systems 7 (6) (1992) 26–33.
[75] M.H. Sokolsky, G.E. Kaiser, A framework for immigrating existing software into new software development environments, Software Engineering Journal 6 (6) (1991) 435–453.
[76] B. Peuschel, W. Schäfer, Concepts and implementation of a rule-based process engine, in: T. Montgomery (Ed.), Proceedings of the 14th International Conference on Software Engineering, ACM Press, Boston, 1992.
[77] V. Ambriola, P. Ciancarini, C. Montangero, Software process enactment in oikos, ACM SIGSOFT Software Engineering Notes 15 (6) (1990) 183–192.
[78] I.Z. Ben-Shaul, A paradigm for decentralized process modeling and its realization in the oz environment, PhD Thesis, Columbia University, Department of Computer Science, a revised version appears as Israel Ben-Shaul and Gail E. Kaiser, A Paradigm for Decentralized Process Modeling, Kluwer Academic Publishers, Boston MA, 1995 (April 1995).
[79] D. Heimbigner, The processwall: a process state server approach to process programming, in: Proceedings of the Fifth ACM/SIGSOFT Conference on Software Development Environments, Washington, DC, 1992, pp. 159–168.
[80] D. Bronisz, C. Fernstrom, Process support in practice: a case study, in: Proceedings of the Software Engineering Environments Conference, 1993, pp. 24–34.
[81] G. Cugola, E. Di Nitto, C. Ghezzi, M. Mantione, How to deal with deviations during process model enactment, in: D. Perry (Ed.), Proceedings of the 17th International Conference on Software Engineering, ACM Press, Seattle, Washington, 1995, pp. 55–62.
[82] S. Bandinelli, M. Braga, A. Fuggetta, L. Lavazza, Cooperation support in the spade environment: a case study, in: Proceedings of the Workshop on Computer Supported Cooperative Work, Petri nets, and Related Formalisms (14th International Conference on Application and Theory of Petri Nets), Chicago, 1993.
[83] E. Di Nitto, A. Fuggetta, Integrating process technology and cscw, in: Fourth European Workshop on Software Process Technology, Leiden, The Nederland, 1995.
[84] Digital Equipment Corporation, Dec fuse home page, 1999. http://www.digital.com/fuse/
[85] Sun Microsystems, Tooltalk user's guide, August 1997. ftp://192.18.99.138/802-7318/802-7318.pdf